Infinite Time Cellular Automata: a Real Computation Model

Fabien Givors, Grégory Lafitte, Nicolas Ollinger

Journées Automates Cellulaires 2010 (Turku), pp. 111-120

INFINITE TIME CELLULAR AUTOMATA: A REAL COMPUTATION MODEL

FABIEN GIVORS, GREGORY LAFITTE, AND NICOLAS OLLINGER

Laboratoire d'Informatique Fondamentale de Marseille (LIF), CNRS – Aix-Marseille Université, 39 rue Joliot-Curie, 13453 Marseille Cedex 13, France

Abstract. We define a new transfinite time model of computation, *infinite time cellular automata*. The model is shown to be as powerful than infinite time Turing machines, both on finite and infinite inputs; thus inheriting many of its properties. We then show how to simulate the canonical real computation model, *BSS machines*, with infinite time cellular automata in exactly ω steps.

Introduction

When the second and third authors of this paper were in their PhD years, their common advisor, Jacques Mazoyer, had encouraged them to define a computation model on real numbers using cellular automata. The second author had defined at the end of the Nineties a cellular automata generalization running into transfinite time, but it was never published and was only clumsily defined in his PhD thesis. They thought of using it as a real computation model in 2001 in Riga but never did anything about it since. This paper is a description of these ideas which had stayed for ten years in the state of scribbled notes.

Transfinite time computation models were first considered in 1989 by Hamkins and Kidder. Hamkins and Lewis later developed the model of infinite time Turing machines and its theory in [HL00]. Koepke [Koe06] defined another transfinite time model based on register machines, that was later refined by Koepke and Miller [KM08]. These models differ in their computation power, the infinite time Turing machines model being the most powerful one.

In [BSS89], Blum, Shub and Smale introduced, also in 1989, a model of computation, coined *BSS machines*, intended to describe computations over the real numbers. It can be viewed as Turing machines with tapes whose cells (or Random Access Machines with registers that) can store arbitrary real numbers and that can compute rational functions overs reals at unit cost. Their model, which is more general[1] than the presentation provided in this paper, is a canonical model of real computation. The idea of real computation is to deal with hypothetical computing

The research presented in this paper has been made possible by the support of the French ANR grants *NAFIT* (ANR-08-DEFIS-008-01) and *EMC* (ANR-09-BLAN-0164-01).

[1]It really provides a setting for computing over rather arbitrary structures.

machines using infinite-precision real numbers and to be able to prove results of computations operating on the set of real numbers. A typical example is studying the computability of the Mandelbrot set. It can be viewed as an idealized analog computer which operates on real numbers and is differential, whereas digital computers are limited to integers and are algebraic. For a survey of analog computations, the reader is referred to the survey [BC08].

In this paper, we introduce a transfinite time computation model, the *infinite time cellular automata*. The model is arguably more *natural* and *uniform* than other transfinite time models introduced, for the same reasons cellular automata are more *natural* and *uniform* than Turing machines. There is no head wandering here and there and there is no difference between states and data.

We show that the infinite time cellular automata have the same computing power than infinite time Turing machines, both on finite and infinite inputs. They thus inherit the nice properties of this latter model. We then show how to simulate the BSS machines with infinite time cellular automata in exactly ω steps. We finish by introducing another transfinite time model based on cellular automata.

1. Infinite time cellular automata

Definition 1.1. An *infinite time cellular automaton* (ITCA) A is defined by Σ, the finite set of states of A, linearly ordered by $\prec$ and with a least element $\mathbf{0}$; and $\delta : \Sigma^3 \to \Sigma$, the local rule of A, satisfying $\delta(\mathbf{0}, \mathbf{0}, \mathbf{0}) = \mathbf{0}$, so that $\mathbf{0}$ is a *quiescent* state.

A *configuration* is an element of $\Sigma^{\mathbb{Z}}$. The local rule δ induces a global rule $\Delta : \Sigma^{\mathbb{Z}} \to \Sigma^{\mathbb{Z}}$ on configurations such that $\Delta(C)_i = \delta(C_{i-1}, C_i, C_{i+1})$ for $i \in \mathbb{Z}$ and $C \in \Sigma^{\mathbb{Z}}$.

Starting from a configuration $C \in \Sigma^{\mathbb{Z}}$, the *evolution* of length $\theta \in \mathrm{Ord}$ of A is given by $(\Delta^\alpha(C))_{\alpha \leqslant \theta}$:

$$\begin{array}{rcl} \Delta^{\beta+1}(C) & = & \Delta(\Delta^\beta(C)) \\ \Delta^\lambda(C)_i & = & \liminf^{\prec}_{\gamma<\lambda} \Delta^\gamma(C)_i \text{ for all } i \in \mathbb{Z} \text{ and } \lambda \text{ limit} \end{array}$$

Definition 1.2. Let $\mathbf{h} \in \Sigma$ a particular state we will refer to as the *halting* state.

An evolution of length θ is called a *computation* if the state $\mathbf{h}$ appears in the last configuration, $\Delta^\theta(C)$, but not before this stage.

We settle a convention on the way we code integers or real numbers in our model. For example, we could code integers and real numbers in binary (using $\mathbf{0}$ and another state as symbols for 0 and 1) on the right cells (cells whose indices belong to $\mathbb{N}$).

Definition 1.3. Let X be a space for which a coding has been settled. (For example, $\mathbb{N}$, $\mathbb{R}$, $2^{\mathbb{N}}$ or $2^{\mathbb{Z}}$.)

A (partial) function F on X, $F : X \to X$, is said to be *infinite time computable* if there is an infinite time cellular automaton such that for each $x \in dom(F)$, there is a computation starting with a configuration with a coding of x that halts on a configuration with a similar coding of $F(x)$.

A set $A \subseteq X$ is *infinite time decidable* if its characteristic function is infinite time computable, and is *infinite time semi-decidable* if it is the domain of an infinite time computable function.

We use the term "semi-decidable" instead of "enumerable", since contrarily to the classical computability concepts, in the transfinite time context, being semi-decidable is not equivalent to being the range of a computable function.

2. Properties of infinite time cellular automata

2.1. Comparisons with other infinite time models

Hamkins, Kidder and Lewis [HL00] have defined an infinite time Turing machines model and Koepke [Koe06] has defined an infinite time register machines model, that was later refined by Koepke and Miller [KM08].

The *infinite time Turing machines* (ITTM) work as a classical Turing machine with a head reading and writing on a bi-infinite tape, moving left and right in accordance with the instructions of a finite program with finitely many states. At successor stages of computation, the machine operates in exactly the classical manner. At limit stages, the machine enters a special limit state, with the head on the origin cell, and each cell of the tape taking the value of the lim inf of the values appearing in that cell before that limit stage.

The *infinite time register machines* behave like standard register machines at successor stages. At limit times, the register contents are defined using lim inf's of the previous register contents. The difficulty here is that the lim inf does not necessarily exist in that case, since a register can contain arbitrary large integers. The machines of [Koe06] crashed in such a case. Those of [KM08] continue beyond such crashes by resetting a register to 0 whenever it overflows.

The infinite time Turing machines are strictly stronger than infinite time register machines: the halting problem for infinite time register machines can be decided by an ITTM.

Theorem 2.1. *Infinite time cellular automata have the same computing power of infinite time Turing machines.*

Proof. The right to left implication goes as follows:

At successor stages, the simulation of ITTM by ITCA works the same way it does in the non-infinite-time case.

At limit stages, we have to put the configuration of the ITCA in the limit configuration simulation of the ITTM simulated. To do this, we need to be able to know that we are at a limit stage. It suffices to have two adjacent cells of the ITCA at the origin that, at successor stages, alternate between **0** and some other state such that the adjacent states are different. At the next limit stage, they will be both equal to **0**. We also have to use the same trick on the cells visited by the head to make sure that at limit stages, if the ITTM becomes stationary, we can wipe out the stationary state from our simulation tape and enter in the special limit state. The ITCA can then prepare the configuration to continue the ITTM simulation.

The left to right implication: it takes ω steps with an ITTM machine to simulate an ITCA global step (on an infinite input). It is just then a matter of determining whether the ITTM is at a limit stage or not. This is easily achieved by an ITTM since it enters a special limit state at limit stages. ∎

2.2. Features of those infinite time models

Hamkins, Kidder, Lewis and Welch [HL00, Wel99, Wel00b, Wel00a] have shown many properties of infinite time Turing machines. By Theorem 2.1, infinite time cellular automata have many of these same properties. We state in the following the properties inherited by infinite time cellular automata.

Theorem 2.2. *The set of reals coding well-orders is infinite time decidable.*

The hyperarithmetic sets are those that are decidable in time less than some recursive ordinal. Every Π^1_1 *set is decidable and the class of decidable sets is contained in* Δ^1_2.

Definition 2.3. An ordinal α is *clockable* if there is an ITCA computation starting from the all-but-one quiescent configuration C ($C_0 \neq \mathbf{0}$ and $C_i = \mathbf{0}\ \forall i \in \mathbb{Z} \setminus \{0\}$) and that halts after exactly α steps (meaning that the α^{th} configuration, $\Delta^\alpha(C)$, is the first configuration in which the halting state **h** appears).

A real r is *writable* if it is the output of an ITCA computation. An ordinal is writable if it is coded by such a real.

There are of course only countably many clockable and writable ordinals, since there are only countably many local rules.

Theorem 2.4. *Every recursive ordinal is clockable. Even* $\omega_1^{CK} + \omega$ *is clockable. Beyond that, there are many intervals of non-clockable ordinals. The supremum of clockable ordinals is recursively inaccessible*[2]*. Moreover, the writable ordinals however form an initial segment of the ordinals. The supremum of the writable ordinals is the supremum of the clockable ordinals.*

One of the beautiful theorems of ITTMs that carry through to ITCAs is the Lost Melody Theorem. The real constructed in this theorem is like a lost melody that you can recognize when someones hums it to you, but which you cannot sing on your own.

Theorem 2.5 (Lost Melody Theorem)**.** *There is a real* r *which is recognizable (*$\{r\}$ *is decidable), but not writable.*

There are different ways to construct such lost melody reals. One way is to consider the supremum γ of the ordinal stages by which an ITTM computation, on an empty input, either halts or repeats. Notice that by a simple cofinality argument, we can show that all these ordinals are countable. There is a smallest ordinal $\delta \geqslant \gamma$ such that $L_{\delta+1} \models$ "δ is countable". $L_{\delta+1}$ has a canonical well-ordering, thus there is some real $r \in L_{\delta+1}$ which is least with respect to the canonical L order, such that r codes δ. γ (and thus r) are somehow a generalization of the busy beaver problem to transfinite time computations. It is then not surprising that r cannot be computable, since that would render the infinite time halting problem decidable. It is recognizable because it is possible to reconstruct the L hierarchy using an ITTM and verify that r is really the least coding of an ordinal having the properties of δ.

[2]A *recursively inaccessible* ordinal is an ordinal that is both admissible and a limit of admissibles. An ordinal α is *admissible* if the construction of the Gödel universe, L, up to a stage α, yields a model L_α of Kripke-Platek set theory. The Church-Kleene ordinal, ω_1^{CK}, is the smallest non-recursive ordinal and is the smallest admissible ordinal. For more on admissibles, see the most excellent book of Barwise [Bar75].

3. Computations on the reals

3.1. Blum-Shub-Smale model

Blum, Shub and Smale [BSS89] introduced the *BSS model*.

A simplified presentation of the BSS model goes through defining "Turing machines with real numbers". We follow Hainry's presentation in his PhD thesis [Hai06].

Definition 3.1. A *simplified BSS machine* (or shortly, BSS machine) is composed of an infinite tape and a program. The infinite tape is made of cells, each containing a real number. We denote the tape by $(x_n)_{n\in\mathbb{Z}} \in \mathbb{R}^{\mathbb{Z}}$. The program is a numbered (finite) sequence of instructions. The number of each instruction in the program sequence is seen as a state ($\in Q$). The instructions are

- *go right*: changes the tape to $(x_{n+1})_{n\in\mathbb{Z}}$;
- *go left*: changes the tape to $(x_{n-1})_{n\in\mathbb{Z}}$;
- *branch if greater than* 0: if the current cell (x_0) is greater than 0, then it branches to a specified location in the program;
- *branch if equal to* 0: if the current cell (x_0) is equal to 0, then it branches to a specified location in the program;
- *make a computation*: the current cell (x_0) is changed to be equal to the result of a computation from x_0, x_1 and possible constants ($k \in \mathbb{R}$). A computation is one of the following:
  - $x_0 \leftarrow -x_0$;
  - $x_0 \leftarrow k$;
  - $x_0 \leftarrow x_0 + x_1$;
  - $x_0 \leftarrow x_0 \times x_1$;
  - $x_0 \leftarrow k \times x_0$.

The machine starts by executing the first instruction (with the least number) of the program and continues by executing the next instruction and so on until it branches. It halts when it has no more instructions to execute.

It is possible to give a definition for a more general BSS machine on rather arbitrary structures. In this paper, we will stick with simplified BSS machines on $\mathbb{R}$.

For a lot more on the *BSS model* and computation on the real numbers, the reader is referred to the book by Blum, Cucker, Shub and Smale [BCSS98].

3.2. BSS by ω-ITCA

We show how to simulate a simplified BSS machine with an ITCA.

Theorem 3.2. *A simplified BSS machine can be simulated by an ITCA in ω steps.*

Proof. Consider a BSS machine. At each time step t of a computation, only a finite number of non-zero real numbers are defined: k constants inside the program and $l \leqslant t$ cells of the tape. For the sake of clarity, suppose that the BSS machine works on reals in the interval $[-1, 1]$[3]. Imagine that we encode these $k + l$ reals as

[3]The construction extends to $\mathbb{R}$ at the cost of non significant tricks, for example by adding just after the bit of sign the encoding of the integer part of each real on a same number of bits followed by a dot.

infinite words on the alphabet $\{0, 1\}$ encoding a bit of sign followed by their binary expansion.

Each computation instruction of the BSS machine has the nice property that it can be computed, in time ω, by a Turing machine working synchronously on the representation of its operands. For each finite initial portion of the tape, it is left untouched by such a machine after some finite time. Moreover, at the cost of some extra bookkeeping on the tape, such a computation can be achieved in a reversible way[4].

Each branch instruction of the BSS machine can be achieved in a similar way: one can choose a initial hypothesis on the branch[5] and start a computation by a Turing machine working synchronously on the representation of the operands ; either the machine eventually halts contradicting the hypothesis, or its head moves infinitely towards the end of the infinite words. For each finite initial portion of the tape, it is left untouched by such a machine after some finite time and the computation can be achieved in a reversible way.

Packing it all together, we can simulate a BSS machine if we can launch as many Turing computation threads as needed. Encode the $k + l$ reals encodings as a single infinite word and put some finite control at the beginning of it. The finite control plays the role of the head of the BSS machine, keeping the current state (instruction number). The control is responsible for launching the next instruction: each time there is enough room on its right, it starts a new thread for the current instruction. If the instruction is a move instruction, the thread simply selects the next cell, adding a new 0 real to the tuple if necessary. If the instruction is a branch instruction, the control takes the default hypothesis on the result and launches the branch thread described before. If the instruction is a computation instruction, the control launches the thread described above. At each time step, the control is pointing to a current instruction and a finite number of threads are computing on its right. Each thread works on a finite portion of tape where it should have exclusive access. Somewhere on its right is the last point where he modified the reals. When a thread wants to access a portion of the reals already modified by the next thread on its left, it enforce this thread to undo its computation to restore the reals as they were before. A cascade of undoing occurs in such a situation, each thread undoing the next thread. At some point an undone thread reaches the control and forces the control part to start again from the instruction where this thread was started, removing the thread from the computing area. As each thread eventually goes to infinity, such an inefficient compute/uncompute dance converges. The same applies when a branch thread discovers that its hypothesis was wrong: it goes back to control to take the other branch of computation, forcing threads newer than him to undo. Notice that a key point of the construction is that there is always enough room for bookkeeping: if a thread needs more space, it just can wait for more space to be available before continuing its computation, either it will eventually happen, or a backtrack will occur that can be handled.

The result of a computation is obtained easily: at time ω, if the control eventually converged to an accepting state, the control part encodes an accepting state and the encoded reals contain the result of the computation ; if the BSS machine did not converge, the control part does not encode an accepting state.

[4]Just store the non injective choices on a stack that the head pushes in front of itself.

[5]For branch if equal to zero the hypothesis is that $x_0 = 0$.

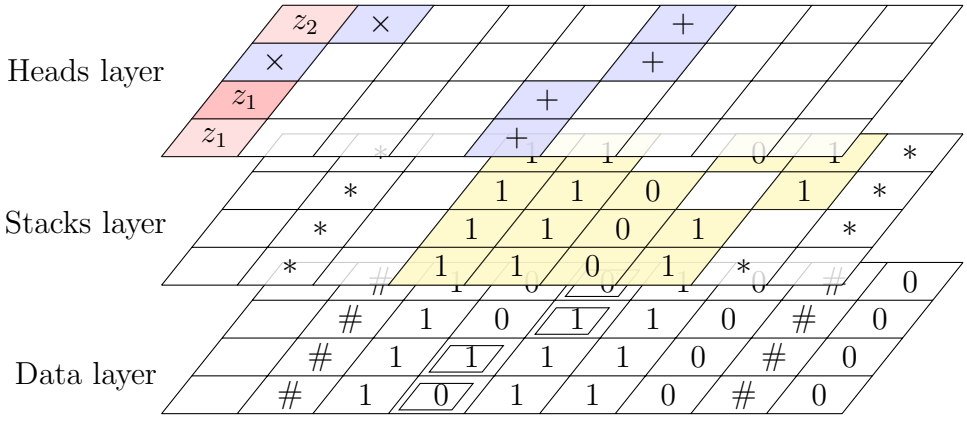


FIGURE 1. Layers encoding computation

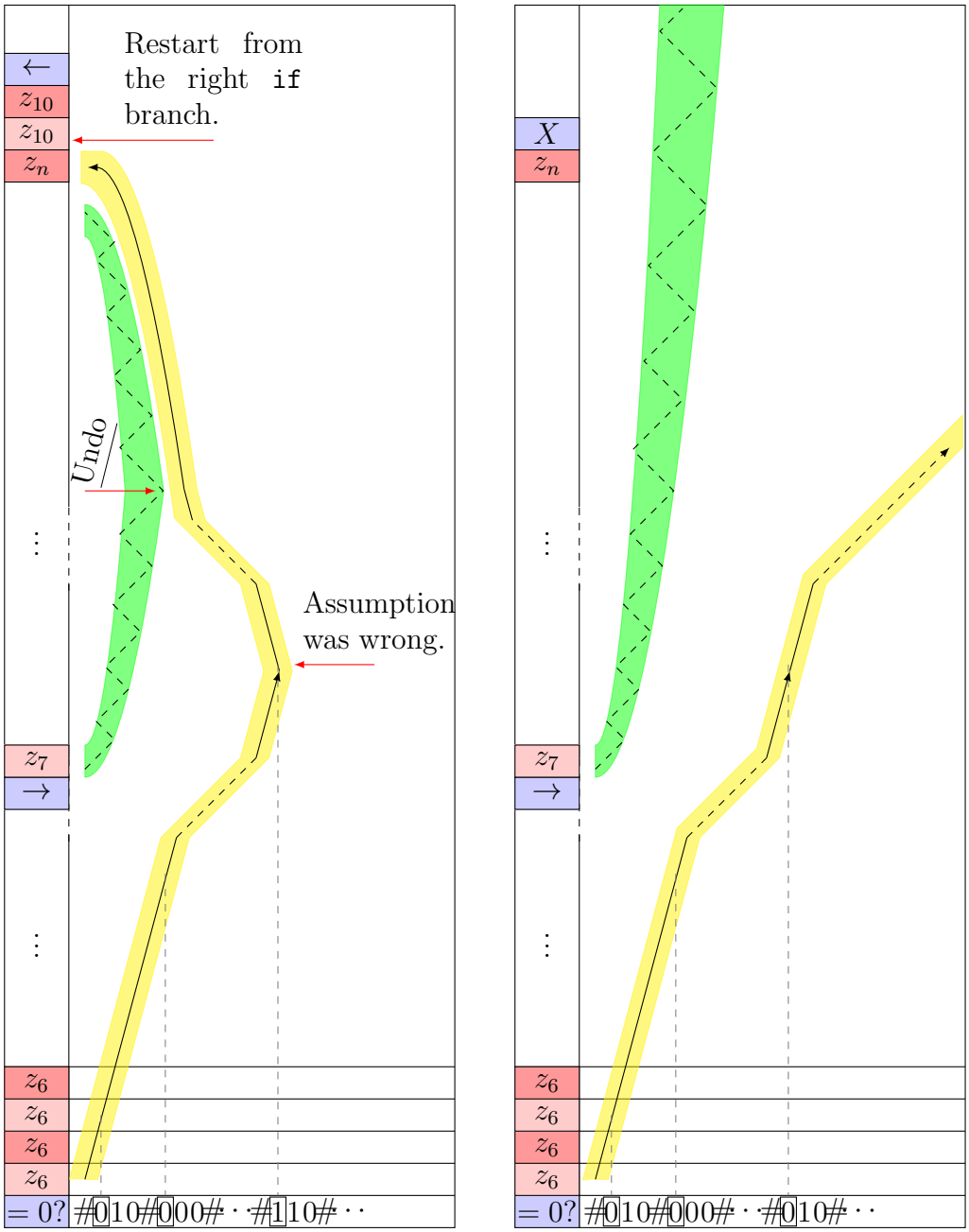
Let us now explain how the described simulation can be carried on a ITCA. Given a BSS machine, the ITCA is constructed as follows. Only a semi-infinite part of the configuration is used. Cell #0 encodes control and the cells on its right encode the reals, the computation area. The computation area is constructed in 3 layers, as depicted on figure 1. The data layer encodes the bits of real numbers and the current position of the head: it is divided into blocks of length $k+l$, separated by # border symbols, containing a bit for each real, the current position of the head being circled. The stack layer encodes the working area for the threads, where each head has its computing stack, and the boundaries of the threads: each thread delimits monotonically the area it already modified by a $*$ symbol. The head layer encodes the heads of the threads, the active parts of the ITCA.

Each instruction of the BSS machine is simulated as explained before. A move thread simply moves the circle to the previous or next bit, adding a new real if necessary, using its thread stack. A branch thread does not modify reals but checks if the hypothesis was true or false, as depicted on figure 2. A computation thread modifies the reals, making choices (for example the values of carries when adding two reals), backtracking when the choice was a bad one, as depicted for addition on figure 3a. Notice that multiplication can be significantly simplified by the fact that we can create new threads, thus it can be decomposed into additions launched by a master thread, as depicted on figure 3b.

It is important to notice that the monotonicity of the forward movement of $*$ symbols ensures that there is no concurrency problem due to neighbor threads backtracking and coming back forward in a same area: when a thread wants to access an area already explored by its follower, the follower is asked to undo its computation. To ask a neighbor to undo, a thread simply modifies its neighbor $*$ symbol to inform him.

The details of the construction use rather classical but tedious CA encoding tricks. The key argument of the proof is that an ITCA can simulate in time ω the work of an unbounded number of Turing heads, thus achieving the same quantity of work than a ITTM in time ω^2. ∎

By diagonalization, it is easy to see that there are functions on the real numbers, computable by ITCAs in ω steps but that are not computable by BSS machines.



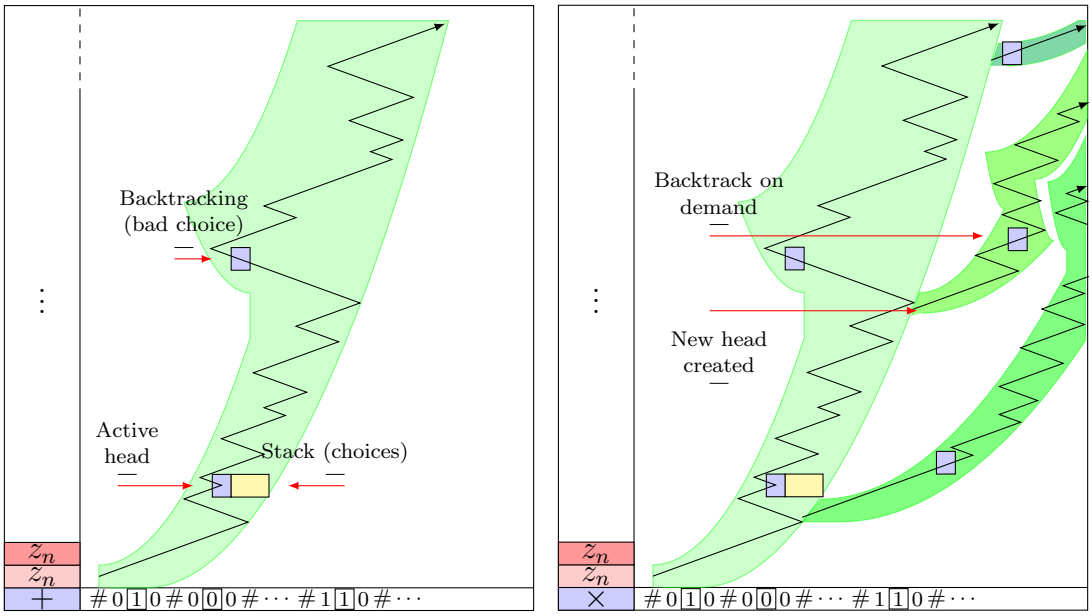
(A) Case $x_0 \neq 0$ (B) Case $x_0 = 0$

FIGURE 2. *Branch if equal to 0* thread

Concluding remarks

We finish this paper by pointing in a direction that we believe to be promising.

Koepke [Koe05] has defined a transfinite time computation model based on Turing machines that has transfinite space. It can thus compute on arbitrary ordinals and sets of ordinals. Koepke and Siders [KS08] have also extended the infinite time register machines model to machines with registers containing arbitrary ordinals. These models make it possible to compute the bounded truth predicate of the constructible universe, $\{(\alpha, \varphi, \vec{x}) : \alpha \in \mathrm{Ord}, \varphi \text{ an } \in\text{-formula}, \vec{x} \in L_\alpha, L_\alpha \models \varphi(\vec{x})\}$, and



(A) Addition thread (B) Multiplication thread

FIGURE 3. Computation threads

allows the following characterization of computable sets of ordinals: a set of ordinals is ordinal computable from of a finite set of ordinal parameters if and only if it an element of Gödel's constructible universe L. Ordinal computability is moreover interesting to be able to reprove some facts about L.

We propose the following definition for ordinal computations over cellular automata.

Definition 3.3. An *ordinal cellular automaton* A is defined by Σ, the finite set of states of A, linearly ordered by $\prec$ and with a least element $\mathbf{0}$; and $\delta : \Sigma^3 \to \Sigma$, the local rule of A, satisfying $\delta(\mathbf{0}, \mathbf{0}, \mathbf{0}) = \mathbf{0}$, so that $\mathbf{0}$ is a *quiescent* state.

A *configuration* of length ξ is an element of Σ^ξ. The local rule δ induces on configurations of length ξ a global rule $\Delta : \Sigma^\xi \to \Sigma^\xi$ such that

$$\begin{cases} \Delta(C)_0 = \delta(\liminf^{\prec}_{\gamma<\xi} C_\gamma, C_0, C_1), \\ \Delta(C)_{\beta+1} = \delta(C_\beta, C_{\beta+1}, C_{\beta+2}), \\ \Delta(C)_\lambda = \delta(\liminf^{\prec}_{\gamma<\lambda} C_\gamma, C_\lambda, C_{\lambda+1}) \quad \text{if } \lambda \text{ limit and } \lambda > 0. \end{cases}$$

Starting from a configuration $C \in \Sigma^\xi$, the *evolution* of length θ of A is given by $(\Delta^\alpha(C))_{\alpha\leqslant\theta}$:

$$\begin{aligned} \Delta^{\beta+1}(C) &= \Delta(\Delta^\beta(C)) \\ \Delta^\lambda(C)_\iota &= \liminf^{\prec}_{\gamma<\lambda} \Delta^\gamma(C)_\iota \text{ for all } \iota < \xi \text{ and } \lambda \text{ limit} \end{aligned}$$

We think that it would be interesting to try to carry the results of the other ordinal machines models over to this ordinal cellular automata model. In this model, there is the limitation due to the fixed length of configurations, which is imposed by the cylindrical nature of our model. There are certainly other ways to allow information to flow from the right to the left of the configurations (and not only left to right).

References

[Bar75] Jon Barwise. *Admissible Sets and Structures: An Approach to Definability Theory*, volume 7 of *Perspectives in Mathematical Logic*. Springer Verlag, 1975.

[BC08] Olivier Bournez and Manuel L. Campagnolo. *New Computational Paradigms. Changing Conceptions of What is Computable*, chapter A Survey on Continuous Time Computations, pages 383–423. Springer-Verlag, New York, 2008.

[BCSS98] Lenore Blum, Felipe Cucker, Mike Shub, and Steve Smale. *Complexity and real computation*. Springer, 1998.

[BSS89] Lenore Blum, Mike Shub, and Steve Smale. On a theory of computation and complexity over the real numbers: Np-completeness, recursive functions and universal machines. *Bulletin of the American Mathematical Society*, 21(1):1–46, 1989.

[Hai06] Emmanuel Hainry. *Modèles de calcul sur les réels: résultats de comparaisons*. PhD thesis, Institut National Polytechnique de Lorraine, 2006.

[HL00] Joel David Hamkins and Andy Lewis. Infinite time turing machines. *Journal of Symbolic Logic*, 65(2):567–604, 2000.

[KM08] Peter Koepke and Russell Miller. An enhanced theory of infinite time register machines. In Arnold Beckmann, Costas Dimitracopoulos, and Benedikt Löwe, editors, *Logic and Theory of Algorithms, 4th Conference on Computability in Europe, CiE 2008, Athens, Greece, June 15-20, 2008, Proceedings*, volume 5028 of *Lecture Notes in Computer Science*, pages 306–315. Springer, 2008.

[Koe05] Peter Koepke. Turing computations on ordinals. *Bulletin of Symbolic Logic*, 11:377–397, 2005.

[Koe06] Peter Koepke. Infinite time register machines. In Arnold Beckmann, Ulrich Berger, Benedikt Löwe, and John V. Tucker, editors, *Logical Approaches to Computational Barriers, Second Conference on Computability in Europe, CiE 2006, Swansea, UK, June 30-July 5, 2006, Proceedings*, volume 3988 of *Lecture Notes in Computer Science*, pages 257–266. Springer, 2006.

[KS08] Peter Koepke and Ryan Siders. Register computations on ordinals. *Archive for Mathematical Logic*, 47:529–548, 2008.

[Wel99] Philip D. Welch. Minimality arguments in the infinite time turing degrees. In S. B. Cooper and J. K. Truss, editors, *Sets and Proofs, Proceedings of the ASL European Meeting, Logic Colloquium 1997*, volume 258 of *London Mathematical Society Lecture Notes in Mathematics*, pages 425–436. Cambridge University Press, April 1999.

[Wel00a] Philip D. Welch. Eventually infinite time turing machine degrees: Infinite time decidable reals. *Journal of Symbolic Logic*, 65:1193–1203, 2000.

[Wel00b] Philip D. Welch. The lengths of infinite time turing machine computations. *Bulletin of the London Mathematica Society*, 32:129–136, 2000.